An Isabelle Correctness Proof for the Volpano/Smith Security Typing System

Gregor Snelting and Daniel Wasserrab
IPD Snelting
Universität Karlsruhe (TH)

May 28, 2015

Abstract

The Volpano/Smith/Irvine security type systems [2] requires that variables are annotated as high (secret) or low (public), and provides typing rules which guarantee that secret values cannot leak to public output ports. This property of a program is called confidentiality.

For a simple while-language without threads, our proof shows that typeability in the Volpano/Smith system guarantees noninterference. Noninterference means that if two initial states for program execution are low-equivalent, then the final states are low-equivalent as well. This indeed implies that secret values cannot leak to public ports. For more details on noninterference and security typing systems, see [1].

The proof defines an abstract syntax and operational semantics for programs, formalizes noninterference, and then proceeds by rule induction on the operational semantics. The mathematically most intricate part is the treatment of implicit flows. Note that the Volpano/Smith system is not flow-sensitive and thus quite unprecise, resulting in false alarms. However, due to the correctness property, all potential breaks of confidentiality are discovered.
Contents

1 The Language 3
  1.1 Variables and Values ........................................ 3
  1.2 Expressions and Commands ................................... 3
  1.3 State ................................................................ 4
  1.4 Small Step Semantics .......................................... 4

2 Security types 10
  2.1 Security definitions ............................................. 10
  2.2 Lemmas concerning expressions ............................... 11
  2.3 Noninterference definitions .................................... 15
    2.3.1 Low Equivalence ........................................... 15
    2.3.2 Non Interference ........................................... 15

3 Executing the small step semantics 25
  3.1 An example taken from Volpano, Smith, Irvine ............. 27
theory Semantics
imports Main
begin

1 The Language

1.1 Variables and Values
datatype val
  = Bool bool — Boolean value
  | Intg int — integer value

abbreviation true == Bool True
abbreviation false == Bool False

datatype expr
  = Val val — value
  | Var vname — local variable
  | BinOp expr bop expr (- - [80,0,81] 80) — binary operation

Note: we assume that only type correct expressions are regarded as later proofs fail if expressions evaluate to None due to type errors. However there is [yet] no typing system

fun binop :: bop ⇒ val ⇒ val ⇒ val option
where
binop Eq v1 v2 = Some(Bool(v1 = v2))
binop And (Bool b1) (Bool b2) = Some(Bool(b1 ∧ b2))
binop Less (Intg i1) (Intg i2) = Some(Bool(i1 < i2))
binop Add (Intg i1) (Intg i2) = Some(Intg(i1 + i2))
binop Sub (Intg i1) (Intg i2) = Some(Intg(i1 - i2))
binop bop v1 v2 = Some(Intg(0))

1.2 Expressions and Commands
datatype com
  = Skip
  | LAss vname expr (:= - [70,70] 70) — local assignment
  | Seq com com (-:/ - [61,60] 60)
  | Cond expr com com (if '(-') -/ else - [80,79,79] 70)
  | While expr com (while '(-') - [80,79] 70)

fun fv :: expr ⇒ vname set — free variables in an expression
where
1.3 State

type-synonym state = vname \rightarrow val

interpret silently assumes type correct expressions, i.e. no expression evaluates to None

fun interpret :: expr \Rightarrow state \Rightarrow val option ([-] -)
where Val: [Val v] s = Some v
| Var: [Var V] s = s V
| BinOp: [e1 <bop> e2] s = (case [e1] s of None \Rightarrow None
| Some v1 \Rightarrow (case [e2] s of None \Rightarrow None
| Some v2 \Rightarrow binop bop v1 v2))

1.4 Small Step Semantics

inductive red :: com * state \Rightarrow com * state \Rightarrow bool
and red' :: com \Rightarrow state \Rightarrow com \Rightarrow state \Rightarrow bool
(((1 (-/-)) \rightarrow (1 (-/-))) [0,0,0,0] 81)
where
\langle c_1,s_1 \rangle \rightarrow \langle c_2,s_2 \rangle == red (c_1,s_1) (c_2,s_2) |
RedLAss:
\langle V:=e,s \rangle \rightarrow \langle \text{Skip},s(V:=[e] s) \rangle
| SeqRed:
\langle c_1,s \rangle \Rightarrow \langle c_1';s' \rangle \Rightarrow \langle c_1;;c_2,s' \rangle
| RedSeq:
\langle \text{Skip};;c_2,s \rangle \rightarrow \langle c_2,s \rangle
| RedCondTrue:
[\langle b \rangle s = \text{Some true} \Rightarrow \langle \text{if} (b) c_1 \text{ else } c_2,s \rangle \rightarrow \langle c_1,s \rangle
| RedCondFalse:
[\langle b \rangle s = \text{Some false} \Rightarrow \langle \text{if} (b) c_1 \text{ else } c_2,s \rangle \rightarrow \langle c_2,s \rangle
| RedWhileTrue:
[\langle b \rangle s = \text{Some true} \Rightarrow \langle \text{while} (b) c,s \rangle \rightarrow \langle c;;\text{while} (b) c,s \rangle
| RedWhileFalse:
[\langle b \rangle s = \text{Some false} \Rightarrow \langle \text{while} (b) c,s \rangle \rightarrow \langle \text{Skip},s \rangle

lemmas red-induct = red.induct[split-format (complete)]

abbreviation reds :: com \Rightarrow state \Rightarrow com \Rightarrow state \Rightarrow bool
(((1 (-/-)) \rightarrow +/ (1 (-/-))) [0,0,0,0] 81) where
\langle c,s \rangle \Rightarrow +\langle c',s' \rangle == red'' (c,s) (c',s')
lemma Skips: 
\[(\text{Skip},s) \rightarrow^* (c',s') \implies s = s' \land c' = \text{Skip}\]
by (blast elim: converse-rtranclp \text{red}.cases)

lemma LAss-reds: 
\[(V := e,s) \rightarrow^* \langle \text{Skip},s' \rangle \implies s' = s(V := [e] s)\]
proof (induct \(V := e\) \text{ s rule: converse-rtranclp-induct2})
  
  case (step \(s \ c'' \ s''\))
  
  hence \(c'' = \text{Skip}\) and \(s'' = s(V := ([e] s))\) by (auto elim: \text{red}.cases)
  
  with \(\langle c'',s'' \rangle \rightarrow^* \langle \text{Skip},s' \rangle\): show ?case by (auto dest: Skips)
qed

lemma Seq2-reds: 
\[(\text{Skip};c_2,s) \rightarrow^* \langle \text{Skip},s' \rangle \implies \langle c_2,s' \rangle \rightarrow^* \langle \text{Skip},s' \rangle\]
by (induct \(c = \text{Skip};c_2\) \text{ s rule: converse-rtranclp-induct2}) (auto elim: \text{red}.cases)

lemma Seq-reds: 
assumes \(\langle c_1; c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\)
obtains \(s''\) where \(\langle c_1, s \rangle \rightarrow^* \langle \text{Skip}, s'' \rangle\) and \(\langle c_2, s'' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\)
proof -
  
  have \(\exists s''. \langle c_1, s \rangle \rightarrow^* \langle \text{Skip}, s'' \rangle \land \langle c_2, s'' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\)
  
  proof -
    
    \{ fix \(c, c'\)
    
    assume \(\langle c, s \rangle \rightarrow^* \langle c', s' \rangle\) and \(c = c_1;c_2\) and \(c' = \text{Skip}\)
    
    hence \(\exists s''. \langle c_1, s \rangle \rightarrow^* \langle \text{Skip}, s'' \rangle \land \langle c_2, s'' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\)
    
    proof (induct arbitrary: \(c_1\) \text{ rule: converse-rtranclp-induct2})
      
      case refl thus \(\text{thesis}\) by simp
    
    next
      
      case (step \(c, s \ c'' \ s''\))
      
      note IH = \(\langle c'' = c_1;c_2; c' = \text{Skip} \rangle \rightarrow^* \exists sx. \langle c_1, s'' \rangle \rightarrow^* \langle \text{Skip}, sx \rangle \land \langle c_2, sx \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\)
      
      from step
      
      have \(\langle c_1;c_2, s \rangle \rightarrow \langle c'', s'' \rangle\) by simp
      
      hence \(\langle c_1 = \text{Skip} \land c'' = c_2 \land s = s'' \rangle \lor\)
        \(\exists c_1'. \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \land c'' = c_1';c_2\)
      
      by (auto elim: \text{red}.cases)
      
      thus \(\text{thesis}\) by simp
    
    proof
      
      assume \(c_1 = \text{Skip} \land c'' = c_2 \land s = s''\)
      
      with \(\langle c'' = \text{Skip} \rangle\) \(\langle c' = \text{Skip} \rangle\)
      
      show \(\text{thesis}\) by auto
    
    next
      
      assume \(\exists c_1'. \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle \land c'' = c_1';c_2\)
      
      then obtain \(c_1'\) where \(\langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle\) and \(c'' = c_1';c_2\) by blast
      
      from IH \(\langle c'' = c_1';c_2; c' = \text{Skip} \rangle\)
      
      obtain \(sx\) where \(\langle c_1, s'' \rangle \rightarrow^* \langle \text{Skip}, sx \rangle\) and \(\langle c_2, sx \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\)
      
      by blast
      
      from \(\langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle\) \(\langle c_1', s'' \rangle \rightarrow^* \langle \text{Skip}, sx \rangle\):
lemma Seq-red-nE
where red-n-Base
inductive red-n
lemma by
proof
WhileTrue-reds
qed
qed
with ⟨⟨c_1; c_2, s⟩ →∗ ⟨Skip, s'⟩: show ?thesis by auto
qed
qed

lemma Cond-True-or-False:
⟨if (b) c_1 else c_2, s⟩ →∗ ⟨Skip, s'⟩ ⇒ ⟨b⟩ s = Some true ∨ ⟨b⟩ s = Some false
by ⟨(induct c::if (b) c_1 else c_2 s rule: converse-rtranclp-into-rtranclp) (auto elim:red.cases)⟩

lemma CondTrue-reds:
⟨if (b) c_1 else c_2, s⟩ →∗ ⟨Skip, s'⟩ ⇒ ⟨b⟩ s = Some true ⇒ ⟨c_1, s⟩ →∗ ⟨Skip, s'⟩
by ⟨(induct c::if (b) c_1 else c_2 s rule: converse-rtranclp-into-rtranclp) (auto elim:red.cases)⟩

lemma CondFalse-reds:
⟨if (b) c_1 else c_2, s⟩ →∗ ⟨Skip, s'⟩ ⇒ ⟨b⟩ s = Some false ⇒ ⟨c_2, s⟩ →∗ ⟨Skip, s'⟩
by ⟨(induct c::if (b) c_1 else c_2 s rule: converse-rtranclp-into-rtranclp) (auto elim:red.cases)⟩

lemma WhileFalse-reds:
⟨while (b) cx, s⟩ →∗ ⟨Skip, s'⟩ ⇒ ⟨b⟩ s = Some false ⇒ s = s'
proof ⟨induct while (b) cx s rule: converse-rtranclp-into-rtranclp) (auto elim:red.cases) ⟩
case step thus ?case by ⟨(auto elim:red.cases dest: Skip-reds)⟩
qed

lemma WhileTrue-reds:
⟨while (b) cx, s⟩ →∗ ⟨Skip, s'⟩ ⇒ ⟨b⟩ s = Some true
⇒ ⟨exists sz. ⟨cx, s⟩ →∗ ⟨Skip, sz⟩ ∧ ⟨while (b) cx, sz⟩ →∗ ⟨Skip, s'⟩⟩
proof ⟨induct while (b) cx s rule: converse-rtranclp-into-rtranclp) (auto elim:red.cases) ⟩
case step s c'' s'''
hence c'''' = cx''; while (b) cx ∧ s'''' = s by ⟨(auto elim:red.cases)⟩
with ⟨⟨c''', s''''⟩ →∗ ⟨Skip, s'⟩: show ?case by ⟨(auto dest: Seq-reds)⟩⟩
qed

lemma While-True-or-False:
⟨while (b) com, s⟩ →∗ ⟨Skip, s'⟩ ⇒ ⟨b⟩ s = Some true ∨ ⟨b⟩ s = Some false
by ⟨(induct c::while (b) com s rule: converse-rtranclp-into-rtranclp) (auto elim:red.cases)⟩

inductive red-n :: com ⇒ state ⇒ nat ⇒ com ⇒ state ⇒ bool
⟨⟨⟨{⟨⋅,⋅⟩}⟩→∗ {⟨⋅,⋅⟩}⟩ [0, 0, 0, 0, 0] 81⟩
where
red-n-Base: ⟨c, s⟩ → B ⟨c, s⟩
| red-n-Rec: [⟨c, s⟩ → ⟨c'', s''⟩; ⟨c''', s''''⟩ → n ⟨c', s'⟩] ⇒ ⟨c, s⟩ → Suc n ⟨c', s'⟩

lemma Seq-red-nE: assumes ⟨c_1;, c_2, s⟩ → n ⟨Skip, s'⟩
obtains $i \cdot j \cdot s''$ where $\langle c_1, s \rangle \rightarrow^i \langle \text{Skip}, s' \rangle$ and $\langle c_2, s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle$
and $n = i + j + 1$

**proof**

- from $\langle c_1 ; c_2, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle$
- have $\exists i \cdot j \cdot s'' . \langle c_1 , s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land \langle c_2 , s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 1$

**proof** (induct $c_1 ; c_2$ s n $\langle$ $\text{Skip}, s' \rangle$ arbitrary; $c_1$ rule:red-n.induct)
- case (red-n-Rec $s$ $c''$ $s''$ n $s'$)
  - note $IH = (\langle c_1 \rangle \land c'' = c_1 ; c_2$
    - $\implies \exists i \cdot j \cdot s'' . \langle c_1 , s'' \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land \langle c_2 , s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 1$
- from $\langle c_1 ; c_2, s \rangle \rightarrow \langle c'' , s'' \rangle$
- have $(c_1 = \text{Skip} \land c'' = c_2 \land s = s'') \lor$
  - $(\exists c_1' . c'' = c_1 ; c_2 \land \langle c_1 , s \rangle \rightarrow \langle c_1' , s'' \rangle)$
- by (induct $c_1 ; c_2$ - - rule:red-induct) auto

thus ?case

**proof**
- assume $c_1 = \text{Skip} \land c'' = c_2 \land s = s''$
- hence $c_1 = \text{Skip}$ and $c'' = c_2$ and $s = s''$ by simp-all
- from $(c_1 = \text{Skip})$ have $\langle c_1 , s \rangle \rightarrow^0 \langle \text{Skip}, s \rangle$ by (fastforce intro:red-n-Base)
- with $\langle c'' , s'' \rangle \rightarrow^n \langle \text{Skip}, s' \rangle$: $c'' = c_2$: $s = s''$
- show ?thesis by (rule-tac $x = 0$ in exf) auto

next
- assume $\exists c_1' . c'' = c_1' ; c_2 \land \langle c_1 , s \rangle \rightarrow \langle c_1' , s'' \rangle$
- then obtain $c_1'$ where $c'' = c_1' ; c_2$ and $\langle c_1 , s \rangle \rightarrow \langle c_1' , s'' \rangle$ by blast
- from $IH[\langle c'' = c_1' ; c_2 \rangle]$ obtain $i \cdot j \cdot s$
  - where $(\langle c_1' , s'' \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land \langle c_2 , s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle)$
  - and $n = i + j + 1$ by blast
- from $\langle c_1 , s \rangle \rightarrow \langle c_1' , s'' \rangle$: $(\langle c_1' , s'' \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle)$
  - have $\langle c_1 , s \rangle \rightarrow \langle c_1 , s'' \rangle$: $(\langle c_1 , s'' \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle)$ by (rule red-n-red-n-Rec)
- with $\langle c_2 , s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle$: $n = i + j + 1$: show ?thesis
- by (rule-tac $x = \text{Suc} i$ in exf) auto

qed

with that show ?thesis by blast

qed

**lemma** while-red-nE:

$(\langle b , \text{cx} , s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle)$

$\implies (\langle b , \text{cx} , s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle)$

$(\exists i \cdot j \cdot s'' . \langle b , \text{cx} , s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land$

$(\langle b , \text{cx} , s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle)$

**proof** (induct while $b$ $\text{cx} \cdot s$ $\langle$ $\text{Skip}, s' \rangle$ rule:red-n.induct)
- case (red-n-Rec $b$ $\text{cx}$ $s$ $s'$ $s''$)
  - from $(\langle b , \text{cx} , s \rangle \rightarrow \langle c'' , s'' \rangle)$
    - have $\langle b , \text{cx} , s \rangle = \langle c'' , s'' \rangle$ by (fastforce intro:red-n-Base)
    - with $\langle b , \text{cx} , s \rangle \rightarrow \langle c'' , s'' \rangle$: $b = \text{cx}$ and $s'' = s$ by blast
  - by (induct while $b$ $\text{cx} \cdot$ - - rule:red-induct) auto

thus ?case

**proof**
\textbf{assume} \[b\] s = Some false \land c'' = Skip \land s'' = s

\textbf{hence} \[b\] s = Some false and c'' = Skip and s'' = s by simp-all

\textbf{with} \(\langle c'',s'' \rangle \rightarrow^n (\text{Skip},s')\) \textbf{have} s = s' and \(n = 0\)

\textbf{by}(induct - - Skip - rule:red-n.induct,auto elim:red.cases)

\textbf{with} \[b\] s = Some false; \textbf{show} \(?\text{thesis by fastforce}\)

\textbf{next}

\textbf{assume} \[b\] s = Some true \land c'' = cx;\textbf{while} (b) cx \land s'' = s

\textbf{hence} \[b\] s = Some true and c'' = cx;\textbf{while} (b) cx

\textbf{and} s'' = s by simp-all

\textbf{with} \(\langle c'',s'' \rangle \rightarrow^n (\text{Skip},s')\)

\textbf{obtain} \(i\) \(j\) \(x\) \(s\) where \((\langle cx,s \rangle \rightarrow^i (\text{Skip},sx)\) and \((\textbf{while} (b) cx,sx) \rightarrow^j (\text{Skip},s')\)

\textbf{and} \(n = i + j + 1\) \textbf{by}(fastforce elim:Seq-red-nE)

\textbf{with} \[b\] s = Some true; \textbf{show} \(?\text{thesis by fastforce}\)

\textbf{qed}

\textbf{qed}

\textbf{lemma} while-red-n.induct \(\text{[consumes 1, case-names false true]}:\)

\textbf{assumes} major: \(\langle\textbf{while} (b) cx,s\rangle \rightarrow^n (\text{Skip},s')\)

\textbf{and} I_ffalse: \(\textbf{\forall} s.\ [b] s = \text{Some false} \Rightarrow P s s\)

\textbf{and} I_Hfalse: \(\textbf{\forall} i j s''.\ [b] s = \text{Some true;} \langle cx,s \rangle \rightarrow^i (\text{Skip},s'')\)

\(\langle\textbf{while} (b) cx,s''\rangle \rightarrow^j (\text{Skip},s')\; P s s'\)

\textbf{shows} P s s'

\textbf{using} major

\textbf{proof}(induct \(n\) arbitrary; s rule:nat-less-induct)

\textbf{fix} \(n\) \(s\)

\textbf{assume} I_ffalse: \(\forall m<n.\ \forall x.\ \langle\textbf{while} (b) cx,x\rangle \rightarrow^m (\text{Skip},s') \Rightarrow P x s'\)

\textbf{and} \(\langle\textbf{while} (b) cx,s\rangle \rightarrow^n (\text{Skip},s')\)

\textbf{from} \(\langle\textbf{while} (b) cx,s\rangle \rightarrow^n (\text{Skip},s')\)

\textbf{have} \([b] s = \text{Some false} \land s = s' \land n = 1\) \lor

\(\exists i j s''.\ [b] s = \text{Some true;} \langle cx,s \rangle \rightarrow^i (\text{Skip},s'')\)

\textbf{and} \(\langle\textbf{while} (b) cx,s''\rangle \rightarrow^j (\text{Skip},s')\) \land \(n = i + j + 2\)

\textbf{by}(rule while-red-nE)

\textbf{thus} P s s'

\textbf{proof}

\textbf{assume} I_Hfalse: \[b\] s = Some false and s = s' by auto

\textbf{from} I_Hfalse \(\text{OF}[b] s = \text{Some false}\) \textbf{have} P s s

\textbf{with} \(s = s\) \textbf{show} \(?\text{thesis by simp}\)

\textbf{next}

\textbf{assume} \(\exists i j s''.\ [b] s = \text{Some true} \land \langle cx,s \rangle \rightarrow^i (\text{Skip},s'')\)

\textbf{and} \(\langle\textbf{while} (b) cx,s''\rangle \rightarrow^j (\text{Skip},s')\) \land \(n = i + j + 2\)

\textbf{then obtain} \(i j s''\) \textbf{where} \[b\] s = Some true

\textbf{and} \(\langle cx,s \rangle \rightarrow^i (\text{Skip},s'')\) \textbf{and} \(\langle\textbf{while} (b) cx,s''\rangle \rightarrow^j (\text{Skip},s')\)

\textbf{and} \(n = i + j + 2\) \textbf{by} blast

\textbf{with I_Hfalse have} P s'' s'

\textbf{apply} (erule-tac \(x=j\) in allE) \textbf{apply clarsimp done}

\textbf{from} I_Hfalse \(\text{OF}[b] s = \text{Some true;} \langle cx,s \rangle \rightarrow^i (\text{Skip},s'')\)
\langle \text{while} \ (b) \ cx.s' \rangle \rightarrow^\exists \langle \text{Skip},s' \rangle \ \text{this} \ \text{show} \ \text{thesis}.

\begin{proof}
  \text{qed}
\end{proof}

\text{lemma} \ \text{reds-to-red-n} : \langle c,s \rangle \rightarrow^\exists \langle c',s' \rangle \implies \exists \ n. \ \langle c,s \rangle \rightarrow^n \langle c',s' \rangle

\text{by}(\text{induct rule:} \text{converse-rtranclp-induct2}, \text{auto intro:} \text{red-n-intros})

\text{lemma} \ \text{red-n-to-reds} : \langle c,s \rangle \rightarrow^n \langle c',s' \rangle \implies \langle c,s \rangle \rightarrow \langle c',s' \rangle

\text{by}(\text{induct rule:} \text{red-n.induct}, \text{auto intro:} \text{converse-rtranclp-into-rtranclp})

\text{lemma} \ \text{while-reds-induct} [\text{consumes 1, case-names false true}]:

\begin{align*}
\langle \text{while} \ (b) \ cx.s \rangle \rightarrow^\exists \langle \text{Skip},s' \rangle; \ \forall s. \ [b] s = \text{Some false} \implies P \ s \ s; \\
\langle \text{while} \ (b) \ cx.s'' \rangle \rightarrow^\exists \langle \text{Skip},s' \rangle; \ \langle cx.s \rangle \rightarrow^\exists \langle \text{Skip},s'' \rangle; \ \langle cx.s'' \rangle \rightarrow P \ s \ s' \\
\end{align*}

\text{apply} (\text{erule} \ \text{reds-to-red-n}, \text{clarsimp})

\text{apply} (\text{erule} \ \text{while-red-n-induct}, \text{clarsimp})

\text{by}(\text{auto dest:} \text{red-n-to-reds})

\text{lemma} \ \text{red-det}:

\begin{align*}
\langle c,s \rangle \rightarrow \langle c_1,s_1 \rangle; \ \langle c,s \rangle \rightarrow \langle c_2,s_2 \rangle \implies c_1 = c_2 \land s_1 = s_2
\end{align*}

\text{proof}(\text{induct arbitrary}; c_2 \ \text{rule:} \text{red-induct})

\text{case} (\text{SeqRed} c_1 s c_1' s_1' c_2')

\text{note} \ \text{IH} = \langle \text{\& c_2.} \ c_1,s \rangle \rightarrow \langle c_2,s_2 \rangle \implies c_1' = c_2 \land s' = s_2

\text{from} \ \langle c_1';c_2',s \rangle \rightarrow \langle c_2,s_2 \rangle \ \text{have} \ c_1 = \text{Skip} \lor (\exists \ cx. \ c_2 = cx.;c_2' \land c_1,s \rightarrow \langle cx.s_2 \rangle)

\text{by} (\text{fastforce elim:} \text{red.cases})

\text{thus} \ ?\text{case}

\text{proof}

\text{assume} \ c_1 = \text{Skip}

\text{with} \ \langle c_1,s \rangle \rightarrow \langle c_1',s' \rangle \ \text{have} \ \text{False} \ \text{by} (\text{fastforce elim:} \text{red.cases})

\text{thus} \ \text{thesis} \ \text{by simp}

\text{next}

\text{assume} \ \exists \ cx. \ c_2 = cx.;c_2' \land (c_1,s) \rightarrow \langle cx.s_2 \rangle

\text{then obtain} \ cx \ \text{where} \ c_2 = cx.;c_2' \ \text{and} \ \langle c_1,s \rangle \rightarrow \langle cx.s_2 \rangle \ \text{by blast}

\text{from} \ \text{IH}[OF \ \langle c_1,s \rangle \rightarrow \langle cx.s_2 \rangle] \ \text{have} \ c_1' = cx \land s' = s_2.

\text{with} \ \langle c_2 = cx.;c_2' \rangle \ \text{show} \ \text{thesis} \ \text{by simp}

\text{qed}

\text{qed} (\text{fastforce elim:} \text{red.cases})+
from \((\text{Skip},s_1) \rightarrow^* (\text{Skip},s_2)\) show ?case
  by \(-(erule \text{converse-rtranclpE}, \text{auto elim:red.cases})\)
next
  case (step \(c'' s'' c' s'
  note \(IH = \langle c'', s'' \rangle \rightarrow^* (\text{Skip},s_2) \Longrightarrow s_1 = s_2\)
  from step have \(\langle c'', s'' \rangle \rightarrow (c', s')\)
    by simp
  from \(\langle c'', s'' \rangle \rightarrow^* (\text{Skip},s_2)\) this have \(\langle c', s' \rangle \rightarrow^* (\text{Skip},s_2)\)
    by \(-(erule \text{converse-rtranclpE}, \text{auto elim:red.cases dest:red-det})\)
  from \(IH\) \([\text{OF this}]\) show ?thesis .
qed

end
theory secTypes
imports Semantics
begin

2 Security types

2.1 Security definitions
datatype secLevel = Low | High

type-synonym typeEnv = vname \rightarrow secLevel

inductive secExprTyping :: typeEnv \Rightarrow expr \Rightarrow secLevel \Rightarrow \text{bool} (-\vdash - : -)
  where typeVal: \(\Gamma \vdash \text{Val } V : \text{lev}\)
    | typeVar: \(\Gamma Vn = \text{Some lev} \Longrightarrow \Gamma \vdash \text{Var } Vn : \text{lev}\)
    | typeBinOp1: \(\Gamma \vdash e1 : \text{Low}; \Gamma \vdash e2 : \text{Low} \Longrightarrow \Gamma \vdash e1 \triangleleft bop \triangleright e2 : \text{Low}\)
    | typeBinOp2: \(\Gamma \vdash e1 : \text{High}; \Gamma \vdash e2 : \text{lev} \Longrightarrow \Gamma \vdash e1 \triangleleft bop \triangleright e2 : \text{High}\)
    | typeBinOp3: \(\Gamma \vdash e1 : \text{lev}; \Gamma \vdash e2 : \text{High} \Longrightarrow \Gamma \vdash e1 \triangleleft bop \triangleright e2 : \text{High}\)

inductive secComTyping :: typeEnv \Rightarrow secLevel \Rightarrow com \Rightarrow \text{bool} (-,-\vdash -)
  where typeSkip: \(\Gamma,T \vdash \text{Skip}\)
    | typeAssH: \(\Gamma V = \text{Some High} \Longrightarrow \Gamma,T \vdash V := e\)
    | typeAssL: \(\Gamma \vdash e : \text{Low}; \Gamma V = \text{Some Low} \Longrightarrow \Gamma,\text{Low} \vdash V := e\)
    | typeSeq: \(\Gamma,T \vdash c1;\Gamma,T \vdash c2 \Longrightarrow \Gamma,T \vdash c1 ; ; c2\)
    | typeWhile: \(\Gamma \vdash b : T; \Gamma,T \vdash c \Longrightarrow \Gamma,T \vdash \text{while } (b) c\)
2.2 Lemmas concerning expressions

**lemma exprTypeable:**

assumes \( \text{fv } e \subseteq \text{dom } \Gamma \) obtains \( T \) where \( \Gamma \vdash e : T \)

**proof** -

from \( \text{fv } e \subseteq \text{dom } \Gamma \) have \( \exists T. \Gamma \vdash e : T \)

**proof (induct \( e \))**

- **case (\( \text{Val } V \))**
  
  have \( \Gamma \vdash \text{Val } V : \text{Low} \) by (rule typeVal)
  
  thus \( \exists \) by (rule exI)

- **next**
  
  **case (\( \text{Var } V \))**
  
  have \( V \in \text{fv } (\text{Var } V) \) by simp
  
  with \( \text{fv } (\text{Var } V) \subseteq \text{dom } \Gamma \) have \( V \in \text{dom } \Gamma \) by simp
  
  then obtain \( T \) where \( \Gamma V = \text{Some } T \) by auto
  
  hence \( \Gamma \vdash \text{Var } V : T \) by (rule typeVar)
  
  thus \( \exists \) by (rule exI)

- **next**
  
  **case (\( \text{BinOp } e_1 \ bop \ e_2 \))**
  
  note \( \text{IH1} = (\text{fv } e_1 \subseteq \text{dom } \Gamma = \Rightarrow \exists T. \Gamma \vdash e_1 : T) \)
  
  note \( \text{IH2} = (\text{fv } e_2 \subseteq \text{dom } \Gamma = \Rightarrow \exists T. \Gamma \vdash e_2 : T) \)
  
  from \( \text{fv } (e_1 \ bop \ e_2) \subseteq \text{dom } \Gamma \) have \( \text{fv } e_1 \subseteq \text{dom } \Gamma \) and \( \text{fv } e_2 \subseteq \text{dom } \Gamma \) by auto
  
  from \( \text{IH1}[\text{OF } (\text{fv } e_1 \subseteq \text{dom } \Gamma)] \) obtain \( T_1 \) where \( \Gamma \vdash e_1 : T_1 \) by auto
  
  from \( \text{IH2}[\text{OF } (\text{fv } e_2 \subseteq \text{dom } \Gamma)] \) obtain \( T_2 \) where \( \Gamma \vdash e_2 : T_2 \) by auto
  
  show \( \exists \)
  
  **proof (cases \( T_1 \))**
  
  - **case Low**
    
    show \( \exists \) \( \text{thesis} \)
  
  **proof (cases \( T_2 \))**
  
  - **case Low**
    
    have \( \Gamma \vdash e_1 \ bop \ e_2 : \text{Low} \) by (simp add: typeBinOp1)
    
    thus \( \exists \) by (rule exI)
  
  - **next**
    
    **case High**
    
    have \( \Gamma \vdash e_1 \ bop \ e_2 : \text{High} \) by (simp add: typeBinOp3)
    
    thus \( \exists \) by (rule exI)
  
  qed

**next**

case \( \text{High} \)

with \( \Gamma \vdash e_1 : T_1 \) \( \Gamma \vdash e_2 : T_2 \)

have \( \Gamma \vdash e_1 \ bop \ e_2 : \text{High} \) by (simp add: typeBinOp2)
thus thesis by (rule exI)
qed
qed
with that show thesis by blast
qed

lemma exprBinopTypeable:
assumes $\Gamma \vdash e_1 \triangleleft \text{bop} \triangleright e_2 : T$
shows $(\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2)$
using assms by(auto elim:secExprTyping.cases)

lemma exprTypingHigh:
assumes $\Gamma \vdash e : T$ and $x \in \operatorname{fv} e$ and $\Gamma \vdash x = \text{Some High}$
shows $\Gamma \vdash e : \text{High}$
using assms
proof (induct e arbitrary:T)
case (Val V) show ?case by (rule typeVal)
next
case (Var V)
from $(x \in \operatorname{fv} (\text{Var V}))$ have $x = V$ by simp
with $(\Gamma \vdash x = \text{Some High})$ show ?case by(simp add:typeVar)
next
case (BinOp $e_1$ bop $e_2$)
note IH1 = $(\forall T. [\Gamma \vdash e_1 : T; x \in \operatorname{fv} e_1; \Gamma \vdash x = \text{Some High}] \Longrightarrow \Gamma \vdash e_1 : \text{High})$

note IH2 = $(\forall T. [\Gamma \vdash e_2 : T; x \in \operatorname{fv} e_2; \Gamma \vdash x = \text{Some High}] \Longrightarrow \Gamma \vdash e_2 : \text{High})$
from $(\Gamma \vdash e_1 \triangleleft \text{bop} \triangleright e_2 : T)$
have $(\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2)$ by (auto intro!:exprBinopTypeable)
then obtain $T_1$ where $\Gamma \vdash e_1 : T_1$ by auto
from $T$ obtain $T_2$ where $\Gamma \vdash e_2 : T_2$ by auto
from $(x \in \operatorname{fv} (e_1 \triangleleft \text{bop} \triangleright e_2))$ have $x \in (\operatorname{fv} e_1 \cup \operatorname{fv} e_2)$ by simp
hence $x \in \operatorname{fv} e_1 \lor x \in \operatorname{fv} e_2$ by auto
thus ?case
proof
assume $x \in \operatorname{fv} e_1$
from IH1[OF $(\Gamma \vdash e_1 : T_1)$ this $(\Gamma \vdash x = \text{Some High})]$ have $\Gamma \vdash e_1 : \text{High}$.
with $\Gamma \vdash e_2 : T_2$ show thesis by(simp add:typeBinOp2)
next
assume $x \in \operatorname{fv} e_2$
from IH2[OF $(\Gamma \vdash e_2 : T_2)$ this $(\Gamma \vdash x = \text{Some High})]$ have $\Gamma \vdash e_2 : \text{High}$.
with $\Gamma \vdash e_1 : T_1$ show thesis by(simp add:typeBinOp3)
qed
qed

lemma exprTypingLow:
assumes $\Gamma \vdash e : \text{Low}$ and $x \in \operatorname{fv} e$ shows $\Gamma \vdash x = \text{Some Low}$
using `assms` proof (induct `e`)  
  case (Val `V`)  
  have `fv (Val `V) = {}` by (rule `FVc`)  
  with `(x ∈ fv (Val `V))` have `False` by auto  
  thus `?thesis` by simp 
next  
  case (Var `V`) 
  from `(x ∈ fv (Var `V))` have `xV = x = V` by simp 
from `(Γ ⊢ Var `V : Low)` have `Γ V = Some Low` by (auto elim:secExprTyping.cases) 
with `xV` show `?thesis` by simp 
next  
  case (BinOp `e1 bop e2`) 
  note `IH1 = ⟨⟨Γ ⊢ e1 : Low; x ∈ fv e1⟩⟩`  
  note `IH2 = ⟨⟨Γ ⊢ e2 : Low; x ∈ fv e2⟩⟩`  
  from `(Γ ⊢ e1 ≪ bop ≫ e2 : Low)` have `Γ ⊢ e1 : Low and Γ ⊢ e2 : Low` by (auto elim:secExprTyping.cases) 
from `(x ∈ fv (e1 ≪ bop ≫ e2))` have `x ∈ fv e1 ∪ fv e2` by (simp add:`FVe`) 
  hence `x ∈ fv e1 ∨ x ∈ fv e2` by auto  
  thus `?case` proof  
    assume `x ∈ fv e1`  
    with `IH1[OF ⟨Γ ⊢ e1 : Low⟩]` show `?thesis` by auto 
next  
  assume `x ∈ fv e2`  
  with `IH2[OF ⟨Γ ⊢ e2 : Low⟩]` show `?thesis` by auto 
qed 
qed 

lemma `typeableFreevars`: 
  assumes `(Γ ⊢ e : T)` shows `fv e ⊆ dom Γ` 
using `assms` proof (induct `e arbitrary:T`)  
  case (Val `V`) 
  have `fv (Val `V) = {}` by (rule `FVc`) 
  thus `?case` by simp 
next  
  case (Var `V`) 
  show `?case` proof  
    fix `x` assume `x ∈ fv (Var `V)`  
    hence `x = V` by simp 
from `(Γ ⊢ Var `V : T)` have `Γ V = Some T` by (auto elim:secExprTyping.cases) 
with `(x = V)` show `x ∈ dom Γ` by auto 
qed 
next  
  case (BinOp `e1 bop e2`) 
  note `IH1 = (⟨⟨T. Γ ⊢ e1 : T ⟩⟩)` 

**Lemma:** exprNotNone

**Assumes:** \( \Gamma \vdash e \colon T \) and \( \text{fv } e \subseteq \text{dom } s \)

**Shows:** \( [e] s \neq \text{None} \)

**Proof:** (induct \( e \) arbitrary: \( \Gamma \ T \ s \))

1. Case: (Val \( v \))
   - Show \( \texttt{?case by(simp add:Val)} \)

2. Next Case: (Var \( V \))
   - Have \( [\text{Var } V] s = s V \) by (simp add:Var)
   - Have \( V \in \text{fv } (\text{Var } V) \) by (auto simp add:FVv)
   - With \( \text{fv } (\text{Var } V) \subseteq \text{dom } s \) have \( V \in \text{dom } s \) by simp
   - Thus \( \texttt{?case by auto} \)

3. Next Case: (BinOp \( e1 \ bop \ e2 \))

   **Note:** IH1 = \( \bigwedge T. \; \Gamma \vdash e1 : T; \text{fv } e1 \subseteq \text{dom } s \) \( \implies \) \( [e1] s \neq \text{None} \)

   **Note:** IH2 = \( \bigwedge T. \; \Gamma \vdash e2 : T; \text{fv } e2 \subseteq \text{dom } s \) \( \implies \) \( [e2] s \neq \text{None} \)

   From \( \Gamma \vdash e1 \ bop \ e2 \colon T \)

   **Have:** \( \exists T1. \; \Gamma \vdash e1 : T1 \wedge (\exists T2. \; \Gamma \vdash e2 : T2) \)

   by (rule exprBinopTypeable)

   Then obtain \( T1 \ T2 \) where \( \Gamma \vdash e1 : T1 \) and \( \Gamma \vdash e2 : T2 \) by blast

   From \( \text{fv } (e1 \ bop \ e2) \subseteq \text{dom } s \) have \( \text{fv } e1 \cup \text{fv } e2 \subseteq \text{dom } s \) by (simp add:FVv)

   Hence \( \text{fv } e1 \subseteq \text{dom } s \) and \( \text{fv } e2 \subseteq \text{dom } s \) by auto

   From \( \text{IH1}[OF \; (\Gamma \vdash e1 ; T1 ; \text{fv } e1 \subseteq \text{dom } s)] \) have \( [e1]s \neq \text{None} \).

   Moreover from \( \text{IH2}[OF \; (\Gamma \vdash e2 ; T2 ; \text{fv } e2 \subseteq \text{dom } s)] \) have \( [e2]s \neq \text{None} \).

   Ultimately show \( \texttt{?case} \)

   - Apply(\texttt{cases bop}) apply auto
   - Apply(\texttt{case-tac } y, \texttt{auto, case-tac } ya, \texttt{auto})
   - done
2.3 Noninterference definitions

2.3.1 Low Equivalence

Low Equivalence is reflexive even if the involved states are undefined. But in non-reflexive situations low variables must be initialized (i.e. \( \in \text{dom state} \)), otherwise the proof will not work. This is not a restriction, but a natural requirement, and could be formalized as part of a standard type system.

Low equivalence is also symmetric and transitiv (see lemmas) hence an equivalence relation.

definition\ lownequiv :: typeEnv \Rightarrow state \Rightarrow state \Rightarrow bool (\Gamma \vdash s1 \approx_L s2)

where
\[ \Gamma \vdash s1 \approx_L s2 \equiv \forall v \in \text{dom } \Gamma. \Gamma v = \text{Some Low} \rightarrow (s1 v = s2 v) \]

lemma\ lownequivReflexive: \Gamma \vdash s1 \approx_L s1

by simp add:lownequiv-def

lemma\ lownequivSymmetric: \Gamma \vdash s1 \approx_L s2 \Rightarrow \Gamma \vdash s2 \approx_L s1

by simp add:lownequiv-def

lemma\ lownequivTransitive:
\[ \forall s1 s2 s1' s2'. (\Gamma \vdash s1 \approx_L s2 \land \langle c,s1 \rangle \rightarrow^* \langle \text{Skip},s1' \rangle \land \langle c,s2 \rangle \rightarrow^* \langle \text{Skip},s2' \rangle) \rightarrow \Gamma \vdash s1' \approx_L s2' \]

by simp add:lownequiv-def

2.3.2 Non Interference

definition\ noninterference :: typeEnv \Rightarrow com \Rightarrow bool

where\ noninterference\ \Gamma\ c\ \equiv
\[ \forall s1 s2 s1' s2'. (\Gamma \vdash s1 \approx_L s2 \land \langle c,s1 \rangle \rightarrow^* \langle \text{Skip},s1' \rangle \land \langle c,s2 \rangle \rightarrow^* \langle \text{Skip},s2' \rangle) \rightarrow \Gamma \vdash s1' \approx_L s2' \]

lemma\ noninterferenceI:
\[ \forall s1 s2 s1' s2'. (\Gamma \vdash s1 \approx_L s2; \langle c,s1 \rangle \rightarrow^* \langle \text{Skip},s1' \rangle; \langle c,s2 \rangle \rightarrow^* \langle \text{Skip},s2' \rangle) \rightarrow \Gamma \vdash s1' \approx_L s2' \]

by(auto simp:noninterference-def)

lemma\ interpretLow:
\[ \text{assumes } \Gamma \vdash s1 \approx_L s2 \text{ and all:} \forall V \in \text{fv } e. \Gamma V = \text{Some Low} \]

shows\ \[ e \] s1 = \[ e \] s2

using all

proof (induct e)
\[ \text{case } (\text{Val } v) \]

show \[?case\] by (simp add: Val)

next
\[ \text{case } (\text{Var } V) \]
have [Var V] s1 = s1 V and [Var V] s2 = s2 V by (auto simp: Var)
have V ∈ fv (Var V) by (simp add: FVv)
from (V ∈ fv (Var V), ∀ X ∈ fv (Var V), Γ X = Some Low) have Γ V = Some Low by simp
with assms have s1 V = s2 V by (auto simp add: lowEquiv-def)
thus ?case by (auto)
next
case (BinOp e1 bop e2)
note IH1 = (∀ V ∈ fv e1. Γ V = Some Low ⇒ [e1] s1 = [e1] s2)
note IH2 = (∀ V ∈ fv e2. Γ V = Some Low ⇒ [e2] s1 = [e2] s2)
from (∀ V ∈ fv (e1 <bop> e2), Γ V = Some Low) have ∀ V ∈ fv e1. Γ V = Some Low
and ∀ V ∈ fv e2. Γ V = Some Low by auto
from IH1[OF (∀ V ∈ fv e1. Γ V = Some Low)] have [e1] s1 = [e1] s2.
moreover
from IH2[OF (∀ V ∈ fv e2. Γ V = Some Low)] have [e2] s1 = [e2] s2.
ultimately show ?case by (cases [e1] s2, auto)
qed

lemma interpretLow2:
assumes Γ ⊢ e : Low and Γ ⊢ s1 ≈L s2 shows [e] s1 = [e] s2
proof
from Γ ⊢ e : Low have fv e ⊆ dom Γ by (auto dest: typeableFreevars)
have ∀ x ∈ fv e. Γ x = Some Low
proof
  fix x assume x ∈ fv e
  with Γ ⊢ e : Low show Γ x = Some Low by (auto intro: exprTypingLow)
qed
with Γ ⊢ s1 ≈L s2 show ?thesis by (rule interpretLow)
qed

lemma assignNIhghlemma:
assumes Γ ⊢ s1 ≈L s2 and Γ V = Some High and s1′ = s1(V := [e] s1)
and s2′ = s2(V := [e] s2)
shows Γ ⊢ s1′ ≈L s2′
proof
{ fix V' assume V' ∈ dom Γ and Γ V' = Some Low
from Γ ⊢ s1 ≈L s2, Γ V' = Some Low, have s1 V' = s2 V'
  by (auto simp add: lowEquiv-def)
have s1′ V' = s2′ V'
proof (cases V' = V)
  case True
  with Γ V' = Some Low, Γ V = Some High have False by simp
  thus ?thesis by simp
next
  case False
  with (s1′ = s1(V := [e] s1)), (s2′ = s2(V := [e] s2))
have \( sl V' = sl' V' \) and \( s2 V' = s2' V' \) by auto
with \( sl V' = s2 V' \) show ?thesis by simp
qed

} thus ?thesis by(auto simp add:lowEquiv-def)
qed

lemma assignNIlowlemma:
assumes \( \Gamma \vdash sl \approx_L s2 \) and \( \Gamma \vdash V = \text{Some Low} \) and \( \Gamma \vdash e : \text{Low} \)
and \( sl' = sl(V := [e] s1) \) and \( s2' = s2(V := [e] s2) \)
shows \( \Gamma \vdash sl' \approx_L s2' \)
proof
  { fix \( V' \) assume \( V' \in \text{dom} \Gamma \) and \( \Gamma \vdash V' = \text{Some Low} \)
  from \( \Gamma \vdash sl \approx_L s2 \) \( \Gamma \vdash V' = \text{Some Low} \)
  have \( sl V' = s2 V' \) by(auto simp add:lowEquiv-def)
  have \( sl V' = s2 V' \)
  proof(cases \( V' = V \))
    case True
    with \( sl' = sl(V := [e] s1) \) \( s2' = s2(V := [e] s2) \)
    have \( sl' V' = [e] sl \) and \( s2' V' = [e] s2 \) by auto
    from \( \Gamma \vdash e : \text{Low} \) \( \Gamma \vdash sl \approx_L s2 \) have \([e] s1 = [e] s2 \)
    by(auto intro:interpretLow2)
    with \( sl' V' = [e] s1 \) \( s2' V' = [e] s2 \) show ?thesis by simp
  next
    case False
    with \( sl' = sl(V := [e] s1) \) \( s2' = s2(V := [e] s2) \)
    have \( sl' V' = sl V' \) and \( s2' V' = s2 V' \) by auto
    with \( False \) \( sl' V' = sl V' \) \( s2' V' = s2 V' \) by simp
    show ?thesis by auto
  qed
} thus ?thesis by(simp add:lowEquiv-def)
qed

Sequential Compositionality is given the status of a theorem because compositionality is no longer valid in case of concurrency

theorem SeqCompositionality:
assumes nonInterference \( \Gamma c1 \) and nonInterference \( \Gamma c2 \)
sends nonInterference \( \Gamma (c1;c2) \)
proof(rule nonInterferenceI)
  fix \( sl s2 sl' s2' \)
  assume \( \Gamma \vdash sl \approx_L s2 \) and \( \langle c1;c2,s1 \rangle \rightarrow* \langle \text{Skip},sl' \rangle \)
  and \( \langle c1;c2,s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)
  from \( \langle c1;c2,s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) obtain \( sl'' \) where \( \langle c1,s1 \rangle \rightarrow* \langle \text{Skip},s1'' \rangle \)
  and \( \langle c2,s1'' \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) by(auto dest:Seq-reds)
  from \( \langle c1;c2,s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \) obtain \( s2'' \) where \( \langle c1,s2 \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \)
and \( \langle c_2, s_2'' \rangle \rightarrow^{*} \langle \text{Skip}, s_2' \rangle \) by (auto dest:Seg-reds)

from \( \Gamma \vdash s_1 \approx_L s_2 \vdash \langle c_1, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_1'' \rangle \vdash \langle c_1, s_2 \rangle \rightarrow^{*} \langle \text{Skip}, s_2'' \rangle \)

nonInterference \( \Gamma \vdash c_1 \)

have \( \Gamma \vdash s_1'' \approx_L s_2'' \) by (auto simp:nonInterference-def)

with \( \langle c_2, s_1' \rangle \rightarrow^{*} \langle \text{Skip}, s_1' \rangle \vdash \langle c_2, s_2' \rangle \rightarrow^{*} \langle \text{Skip}, s_2' \rangle \) (nonInterference \( \Gamma \vdash c_2 \))

show \( \Gamma \vdash s_1' \approx_L s_2' \) by (auto simp:nonInterference-def)

qd

lemma WhileStepInduct:

assumes while: \( \langle \text{while} \ b \ c, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle \)

and body: \( \forall s_2. \ (c, s_1) \rightarrow^{*} \langle \text{Skip}, s_2 \rangle \rightarrow \Gamma \vdash s_1 \approx_L s_2 \) and \( \Gamma, \text{High} \vdash c \)

shows \( \Gamma \vdash s_1 \approx_L s_2 \)

using while

proof (induct rule:while-reds-induct)

case (false s) thus \(?case\) by (auto simp add:lowEquiv-def)

next
case (true s1 s2)

from body[OF \( \langle c, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_3 \rangle \)] have \( \Gamma \vdash s_1 \approx_L s_3 \) by simp

with \( \Gamma \vdash s_3 \approx_L s_2 \) show \(?case\) by (auto intro:lowEquivTransitive)

qd

In case control conditions from if/while are high, the body of an if/while must not change low variables in order to prevent implicit flow. That is, start and end state of any if/while body must be low equivalent.

theorem highBodies:

assumes \( \Gamma, \text{High} \vdash c \) and \( \langle c, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle \)

shows \( \Gamma \vdash s_1 \approx_L s_2 \)

— all intermediate states must be well formed, otherwise the proof does not work for uninitialized variables. Thus it is propagated through the theorem conclusion

using assms

proof (induct c arbitrary:s1 s2 rule:com.induct)

case Skip

from \( \langle \text{Skip}, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle \) have \( s_1 = s_2 \) by (auto dest:Skip-reds)

thus \(?case\) by (simp add:lowEquiv-def)

next
case (LAss \( V \ e \))

from \( \Gamma, \text{High} \vdash V := e \) have \( \Gamma \ V = \text{Some High} \) by (auto elim:secComTyping.cases)

from \( \langle V := e, s_1 \rangle \rightarrow^{*} \langle \text{Skip}, s_2 \rangle \) have \( s_2 = s_1(V := \[ e \] s_1) \) by (auto intro:LAss-reds)

\{ fix \( V' \) assume \( V' \in \text{dom} \ \Gamma \) and \( \Gamma \ V' = \text{Some Low} \)

have \( s_1 V' = s_2 V' \)

proof (cases \( V' = V \))

case True

with \( \Gamma \ V' = \text{Some Low} \) \( \Gamma \ V = \text{Some High} \) have \( \text{False} \) by simp

thus \(?thesis\) by simp

next
case False

with \( s_2 = s_1(V := \[ e \] s_1) \) show \(?thesis\) by simp

next

next

next
qed
}
thus \textit{?case} by (auto simp add: lowEquiv-def)

next
case (Seq c1 c2)
note IH1 = \(\{s1 \Rightarrow s2. [\Gamma.\text{High} \vdash \mathit{c1}; \{c1,s1\} \rightarrow* \{\mathit{skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_L s2\})
note IH2 = \(\{s1 \Rightarrow s2. [\Gamma.\text{High} \vdash \mathit{c2}; \{c2,s1\} \rightarrow* \{\mathit{skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_L s2\})
from \(\Gamma.\text{High} \vdash \mathit{c1} \& \mathit{c2}\) have \(\Gamma.\text{High} \vdash \mathit{c1}\) and \(\Gamma.\text{High} \vdash \mathit{c2}\)
  by (auto elim: secComTyping.cases)
from \(\{\mathit{c1};\mathit{c2},s1\} \rightarrow* \{\mathit{skip},s2\}\)
  have \(\exists s3. \{\mathit{c1},s1\} \rightarrow* \{\mathit{skip},s3\} \& \{\mathit{c2},s3\} \rightarrow* \{\mathit{skip},s2\}\)
  by (auto intro: Seq-reds)
  then obtain s3 where \((c1,s1) \rightarrow* \{\mathit{skip},s3\} \& \{\mathit{c2},s3\} \rightarrow* \{\mathit{skip},s2\}\)
  by auto
from IH1[OF \(\Gamma.\text{High} \vdash \mathit{c1}\); \(\{\mathit{c1},s1\} \rightarrow* \{\mathit{skip},s3\}\)]
  have \(\Gamma \vdash s1 \approx_L s3\)
  by simp
from IH2[OF \(\Gamma.\text{High} \vdash \mathit{c2}\); \(\{\mathit{c2},s3\} \rightarrow* \{\mathit{skip},s2\}\)]
  have \(\Gamma \vdash s3 \approx_L s2\)
  by simp
from \(\Gamma \vdash s1 \approx_L s3\); \(\Gamma \vdash s3 \approx_L s2\) show \textit{?case}
  by (auto intro: lowEquivTransitive)

next
case (Cond b c1 c2)
note IH1 = \(\{s1 \Rightarrow s2. [\Gamma.\text{High} \vdash \mathit{c1}; \{c1,s1\} \rightarrow* \{\mathit{skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_L s2\})
note IH2 = \(\{s1 \Rightarrow s2. [\Gamma.\text{High} \vdash \mathit{c2}; \{c2,s1\} \rightarrow* \{\mathit{skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_L s2\})
from \(\Gamma.\text{High} \vdash \mathit{if} (b) \mathit{c1} \mathit{else} \mathit{c2}\) have \(\Gamma.\text{High} \vdash \mathit{c1}\) and \(\Gamma.\text{High} \vdash \mathit{c2}\)
  by (auto elim: secComTyping.cases)
from \(\{\mathit{if} (b) \mathit{c1} \mathit{else} \mathit{c2},s1\} \rightarrow* \{\mathit{skip},s2\}\)
  have \([\mathit{b}],\mathit{s1} = \text{Some true} \vee [\mathit{b}],\mathit{s1} = \text{Some false}\)
  by (auto dest: Cond-True-or-False)
thus \textit{?case}
proof
  assume \([\mathit{b}],\mathit{s1} = \text{Some true}\)
  with \(\{\mathit{if} (b) \mathit{c1} \mathit{else} \mathit{c2},s1\} \rightarrow* \{\mathit{skip},s2\}\)
  have \(\{\mathit{c1},s1\} \rightarrow* \{\mathit{skip},s2\}\)
  by (auto intro: CondTrue-reds)
  from IH1[OF \(\Gamma.\text{High} \vdash \mathit{c1}\); \textit{this}\]
  show \textit{thesis}.
next
  assume \([\mathit{b}],\mathit{s1} = \text{Some false}\)
  with \(\{\mathit{if} (b) \mathit{c1} \mathit{else} \mathit{c2},s1\} \rightarrow* \{\mathit{skip},s2\}\)
  have \(\{\mathit{c2},s1\} \rightarrow* \{\mathit{skip},s2\}\)
  by (auto intro: CondFalse-reds)
  from IH2[OF \(\Gamma.\text{High} \vdash \mathit{c2}\); \textit{this}\]
  show \textit{thesis}.
qed

next
case (While b c')
note IH = \(\{s1 \Rightarrow s2. [\Gamma.\text{High} \vdash \mathit{c'}; \{c',s1\} \rightarrow* \{\mathit{skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_L s2\})
from \(\Gamma.\text{High} \vdash \mathit{while} (b) \mathit{c'}\) have \(\Gamma.\text{High} \vdash \mathit{c'}\)
  by (auto elim: secComTyping.cases)
from IH[OF \textit{this}]
  have \(\{s1 \Rightarrow s2. [\{c',s1\} \rightarrow* \{\mathit{skip},s2\}] \Rightarrow \Gamma \vdash s1 \approx_L s2\).
  with \(\{\mathit{while} (b) \mathit{c'},s1\} \rightarrow* \{\mathit{skip},s2\}\)
  have \(\Gamma.\text{High} \vdash \mathit{c'}\)
  show \textit{case} by (auto dest: WhileStepInduct)
qed
lemma CondHighCompositionality:
assumes \( \Gamma, \text{High} \vdash (b) \ c1 \text{ else } c2 \)
shows \( \text{nonInterference } \Gamma \ (if \ (b) \ c1 \text{ else } c2) \)
proof (rule nonInterferenceI)
  \begin{itemize}
  \item fix \( s1 \ s2 \ s1' \ s2' \).
  \item assume \( \Gamma \vdash s1 \approx_L s2 \) and \( (if \ (b) \ c1 \text{ else } c2, s1) \rightarrow\approx \langle \text{Skip}, s1' \rangle \)
    and \( (if \ (b) \ c1 \text{ else } c2, s2) \rightarrow\approx \langle \text{Skip}, s2' \rangle \).
  \item show \( \Gamma \vdash s1' \approx_L s2' \).
  \end{itemize}
proof
  \begin{itemize}
  \item from \( \Gamma, \text{High} \vdash (b) \ c1 \text{ else } c2 \); \( (if \ (b) \ c1 \text{ else } c2, s1) \rightarrow\approx \langle \text{Skip}, s1' \rangle \).
    have \( \Gamma \vdash s1 \approx_L s1' \) by (auto dest: highBodies)
  \item from \( \Gamma, \text{High} \vdash (b) \ c1 \text{ else } c2 \); \( (if \ (b) \ c1 \text{ else } c2, s2) \rightarrow\approx \langle \text{Skip}, s2' \rangle \).
    have \( \Gamma \vdash s2 \approx_L s2' \) by (auto dest: highBodies)
  \item with \( \Gamma \vdash s1 \approx_L s2 \) have \( \Gamma \vdash s1 \approx_L s2' \) by (auto intro: lowEquivTransitive)
  \item from \( \Gamma \vdash s1 \approx_L s1' \) have \( \Gamma \vdash s1' \approx_L s1 \) by (auto intro: lowEquivSymmetric)
  \item with \( \Gamma \vdash s1 \approx_L s2' \) show \( \text{?thesis} \) by (auto intro: lowEquivTransitive)
  qed
qed

lemma CondLowCompositionality:
assumes \( \text{nonInterference } \Gamma \ c1 \) and \( \text{nonInterference } \Gamma \ c2 \) and \( \Gamma \vdash b : \text{Low} \)
shows \( \text{nonInterference } \Gamma \ (if \ (b) \ c1 \text{ else } c2) \)
proof (rule nonInterferenceI)
  \begin{itemize}
  \item fix \( s1 \ s2 \ s1' \ s2' \).
  \item assume \( \Gamma \vdash s1 \approx_L s2 \) and \( (if \ (b) \ c1 \text{ else } c2, s1) \rightarrow\approx \langle \text{Skip}, s1' \rangle \)
    and \( (if \ (b) \ c1 \text{ else } c2, s2) \rightarrow\approx \langle \text{Skip}, s2' \rangle \).
  \item from \( \langle if \ (b) \ c1 \text{ else } c2, s1 \rangle \rightarrow\approx \langle \text{Skip}, s1' \rangle \)
    have \( [b] \ s1 = [b] s2 \) by (auto intro: interpretLow2)
  \item from \( [if \ (b) \ c1 \text{ else } c2, s1] \rightarrow\approx \langle \text{Skip}, s1' \rangle \)
    have \( [b] \ s1 = \text{Some false} \) by (auto dest: Cond-True-or-False)
    thus \( \Gamma \vdash s1' \approx_L s2' \)
  \end{itemize}
proof
  \begin{itemize}
  \item assume \( [b] \ s1 = \text{Some true} \)
    with \( [b] s1 = [b] s2 \) have \( [b] s2 = \text{Some true} \) by (auto intro: CondTrue-reds)
  \item from \( [b] s1 = \text{Some true} \); \( [if \ (b) \ c1 \text{ else } c2, s1] \rightarrow\approx \langle \text{Skip}, s1' \rangle \)
    have \( [c1, s1] \rightarrow\approx \langle \text{Skip}, s1' \rangle \) by (auto intro: CondTrue-reds)
  \item from \( [b] s2 = \text{Some true} \); \( [if \ (b) \ c1 \text{ else } c2, s2] \rightarrow\approx \langle \text{Skip}, s2' \rangle \)
    have \( [c1, s2] \rightarrow\approx \langle \text{Skip}, s2' \rangle \) by (auto intro: CondTrue-reds)
  \item with \( [c1, s1] \rightarrow\approx \langle \text{Skip}, s1' \rangle \); \( \Gamma \vdash s1 \approx_L s2 \); \( \text{nonInterference } \Gamma \ c1 \)
    show \( \text{?thesis} \) by (auto simp: nonInterference-def)
  \end{itemize}
next
  \begin{itemize}
  \item assume \( [b] s1 = \text{Some false} \)
    with \( [b] s1 = [b] s2 \) have \( [b] s2 = \text{Some false} \) by (auto intro: CondTrue-reds)
  \item from \( [b] s1 = \text{Some false} \); \( [if \ (b) \ c1 \text{ else } c2, s1] \rightarrow\approx \langle \text{Skip}, s1' \rangle \)
    have \( [c2, s1] \rightarrow\approx \langle \text{Skip}, s1' \rangle \) by (auto intro: CondFalse-reds)
  \item from \( [b] s2 = \text{Some false} \); \( [if \ (b) \ c1 \text{ else } c2, s2] \rightarrow\approx \langle \text{Skip}, s2' \rangle \)
  \end{itemize}
have \((c_2, s_2) \rightarrow (\text{Skip}, s_2')\) by (auto intro: CondFalse-reds)
with \((c_2, s_1) \rightarrow (\text{Skip}, s_1')\) \(\Gamma \vdash s_1 \approx_L s_2\)
show ?thesis by (auto simp: nonInterference_def)
qed

lemma WhileHighCompositionality:
assumes \(\Gamma, \text{High} \vdash \text{while} (b) \ c'\)
shows \(\text{nonInterference} \ \Gamma \ \text{while} (b) \ c'\)
proof (rule nonInterferenceI)
fix \(s_1 \ s_2 \ s_1' \ s_2'\)
assume \(\Gamma \vdash s_1 \approx_L s_2\) and \(\langle \text{while} (b) \ c', s_1 \rangle \rightarrow^* (\text{Skip}, s_1')\)
and \(\langle \text{while} (b) \ c', s_2 \rangle \rightarrow^* (\text{Skip}, s_2')\)
show \(\Gamma \vdash s_1' \approx_L s_2'\)
proof
- from \(\Gamma, \text{High} \vdash \text{while} (b) \ c' \Rightarrow \langle \text{while} (b) \ c', s_1 \rangle \rightarrow^* (\text{Skip}, s_1')\)
  have \(\Gamma \vdash s_1 \approx_L s_1'\) by (auto dest: highBodies)
- from \(\Gamma, \text{High} \vdash \text{while} (b) \ c' \Rightarrow \langle \text{while} (b) \ c', s_2 \rangle \rightarrow^* (\text{Skip}, s_2')\)
  have \(\Gamma \vdash s_2 \approx_L s_2'\) by (auto dest: highBodies)
with \(\Gamma \vdash s_1 \approx_L s_2\) have \(\Gamma \vdash s_1 \approx_L s_2'\) by (auto intro: lowEquivTransitive)
from \(\Gamma \vdash s_1 \approx_L s_1'\) have \(\Gamma \vdash s_1' \approx_L s_1\) by (auto intro: lowEquivSymmetric)
with \(\Gamma \vdash s_1 \approx_L s_2\) show ?thesis by (auto intro: lowEquivTransitive)
qed

lemma WhileLowStepInduct:
assumes while1: \(\langle \text{while} (b) \ c', s_1 \rangle \rightarrow^* (\text{Skip}, s_1')\)
and while2: \(\langle \text{while} (b) \ c', s_2 \rangle \rightarrow^* (\text{Skip}, s_2')\)
and \(\Gamma \vdash b : \text{Low}\)
and body: \(\forall s_1 \ s_1' \ s_2 \ s_2'. \ ([c', s_1] \rightarrow (\text{Skip}, s_1') \land [c', s_2] \rightarrow (\text{Skip}, s_2')\)
\(\Gamma \vdash s_1 \approx_L s_2\) \(\Longrightarrow\) \(\Gamma \vdash s_1' \approx_L s_2'\)
and le: \(\Gamma \vdash s_1 \approx_L s_2\)
shows \(\Gamma \vdash s_1' \approx_L s_2'\)
using while1 le while2
proof (induct arbitrary: s2 rule: while-reds-induct)
case (false s1)
from \(\Gamma \vdash b : \text{Low}\) \(\Gamma \vdash s_1 \approx_L s_2\) have \([b] \ s_1 = [b] \ s_2\) by (auto intro: interpretLow2)
with \([b] \ s_1 = \text{Some false}\) have \([b] \ s_2 = \text{Some false}\) by simp
with \(\langle \text{while} (b) \ c', s_2 \rangle \rightarrow^* (\text{Skip}, s_2')\) have \(s_2 = s_2'\) by (auto intro: WhileFalse-reds)
with \(\Gamma \vdash s_1 \approx_L s_2\) show ?case by auto
next
case (true s1 s1')
note IH = \(\forall s_2''. \ [\Gamma \vdash s_1' \approx_L s_2''] \land [\text{while} (b) \ c', s_2''] \rightarrow^* (\text{Skip}, s_2'')\)
\(\Longrightarrow\) \(\Gamma \vdash s_1' \approx_L s_2'\)
from \(\Gamma \vdash b : \text{Low}\) \(\Gamma \vdash s_1 \approx_L s_2\) have \([b] \ s_1 = [b] \ s_2\) by (auto intro: interpretLow2)
with \([b] \ s_1 = \text{Some true}\) have \([b] \ s_2 = \text{Some true}\) by simp

with \( \langle \text{while} (b) \ c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \) obtain \( s2'' \) where \( \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \) and \( \langle \text{while} (b) \ c',s2'' \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \) by (auto dest: WhileTrue-bests)

from body[OF \( \langle c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \); \( \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)] have \( \Gamma \vdash s1 \approx_L s2 \).
from IH[OF this \( \langle \text{while} (b) \ c',s2'' \rangle \rightarrow* \langle \text{Skip},s2'' \rangle \)] show ?case .
qed

lemma WhileLowCompositionality:
assumes nonInterference \( \Gamma \ c' \) and \( \Gamma \vdash b : \text{Low} \) and \( \Gamma, \text{Low} \vdash c' \)
shows nonInterference \( \Gamma \ (\text{while} (b) \ c') \)

proof (rule nonInterferenceI)
fix \( s1 \ s2 \ s1' \ s2' \)
assume \( \Gamma \vdash s1 \approx_L s2 \) and \( \langle \text{while} (b) \ c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) and \( \langle \text{while} (b) \ c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)
{ fix \( s1 \ s2 \ s1'' \ s2'' \)
assume \( \langle c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) and \( \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \) and \( \Gamma \vdash s1 \approx_L s2' \)
with (nonInterference \( \Gamma \ c' \) have \( \Gamma \vdash s1'' \approx_L s2'' \)
by (auto simp: nonInterference_def)
}

hence \( \wedge s1 \ s2'' \ s1'' \ s2'' ; \langle \langle c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle ; \langle c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle ; \; \Gamma \vdash s1 \approx_L s2 \] \( \implies \) \( \Gamma \vdash s1'' \approx_L s2'' \) by auto

with \( \langle \text{while} (b) \ c',s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \); \( \langle \text{while} (b) \ c',s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)
(\( \Gamma \vdash b : \text{Low} \) \( \Gamma \vdash s1 \approx_L s2 \) show \( \Gamma \vdash s1' \approx_L s2' \)
by (auto intro: WhileLowStepInduct)
qed

and now: the main theorem:

theorem secTypeImpliesNonInterference:
\( \Gamma,T \vdash c \implies \text{nonInterference} \Gamma c \)

proof (induct c arbitrary: \( T \) rule: com.induct)

case \( \text{Skip} \)
show ?case
proof (rule nonInterferenceI)
fix \( s1 \ s2 \ s1' \ s2' \)
assume \( \Gamma \vdash s1 \approx_L s2 \) and \( \langle \text{Skip},s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) and \( \langle \text{Skip},s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \)
from \( \langle \text{Skip},s1 \rangle \rightarrow* \langle \text{Skip},s1' \rangle \) have \( s1 = s1' \) by (auto dest: Skip-bests)
from \( \langle \text{Skip},s2 \rangle \rightarrow* \langle \text{Skip},s2' \rangle \) have \( s2 = s2' \) by (auto dest: Skip-bests)
from \( \Gamma \vdash s1 \approx_L s2 \) and \( \langle s1 \rangle \approx \langle s1' \rangle \) and \( \langle s2 \rangle \approx \langle s2' \rangle \)
show \( \Gamma \vdash s1' \approx_L s2' \) by simp
qed

done

case \( \text{LA}s V e \)
from \( \Gamma,T \vdash \langle V := e \rangle \)
have \( \text{varprem}:(\Gamma \vdash \text{V = Some High}) \lor (\Gamma \vdash \text{V = Some Low} \land \Gamma \vdash e : \text{Low} \land T = \text{Low}) \)
by (auto elim: secComTyping.cases)

22
show ?case

proof (rule nonInterferenceI)
  fix s₁ s₂ s₁' s₂'
  assume Γ ⊢ s₁ ≈₈ s₂ and ⟨V := e, s₁⟩ →* ⟨Skip, s₁'⟩ and ⟨V := e, s₂⟩ →*
  ⟨Skip, s₂'⟩
  from ⟨⟨V := e, s₁⟩ →* ⟨Skip, s₁'⟩⟩ have s₁' = s₁(V := [e] s₁) by (auto intro:LAss-reds)
  from ⟨⟨V := e, s₂⟩ →* ⟨Skip, s₂'⟩⟩ have s₂' = s₂(V := [e] s₂) by (auto intro:LAss-reds)
  from varprem show Γ ⊢ s₁' ≈₈ s₂'
  proof
    assume Γ V = Some High
    with Γ ⊢ s₁ ≈₈ s₂ ⟨s₁' = s₁(V := [e] s₁) ⟩ ⟨s₂' = s₂(V := [e] s₂) ⟩
    show ?thesis by (auto intro:assignNhighlemma)
  next
    assume Γ V = Some Low ∧ Γ ⊢ e : Low ∧ T = Low
    with Γ ⊢ s₁ ≈₈ s₂ ⟨s₁' = s₁(V := [e] s₁) ⟩ ⟨s₂' = s₂(V := [e] s₂) ⟩
    show ?thesis by (auto intro:assignNlowlemma)
  qed
qed

next
case (Seq c₁ c₂)
  note IH₁ = ⟨\∧ T. Γ, T ⊢ c₁ ⟩ nonInterference Γ c₁
  note IH₂ = ⟨\∧ T. Γ, T ⊢ c₂ ⟩ nonInterference Γ c₂
  show ?case
  proof (cases T)
    case High
    with Γ, T ⊢ c₁;;c₂ have Γ, High ⊢ c₁ and Γ, High ⊢ c₂
      by (auto elim:secComTyping_cases)
    from IH₁[OF Γ, High ⊢ c₁] have nonInterference Γ c₁ .
    moreover
    from IH₂[OF Γ, High ⊢ c₂] have nonInterference Γ c₂ .
    ultimately show ?thesis by (auto intro:SeqCompositionality)
  next
    case Low
    with Γ, T ⊢ c₁;;c₂ have ⟨Γ, Low ⊢ c₁ ∧ Γ, Low ⊢ c₂ ⟩ ∨ Γ, High ⊢ c₁;;c₂
      by (auto elim:secComTyping_cases)
    thus ?thesis
      proof
        assume Γ, Low ⊢ c₁ ∧ Γ, Low ⊢ c₂
        hence Γ, Low ⊢ c₁ and Γ, Low ⊢ c₂ by simp-all
        from IH₁[OF Γ, Low ⊢ c₁] have nonInterference Γ c₁ .
        moreover
        from IH₂[OF Γ, Low ⊢ c₂] have nonInterference Γ c₂ .
        ultimately show ?thesis by (auto intro:SeqCompositionality)
      next
        assume Γ, High ⊢ c₁;;c₂
        hence Γ, High ⊢ c₁ and Γ, High ⊢ c₂ by (auto elim:secComTyping_cases)
  qed

23
from IH1[OF Γ,High ⊢ c1] have nonInterference Γ c1 .
moreover
from IH2[OF Γ,High ⊢ c2] have nonInterference Γ c2 .
ultimately show ?thesis  by(auto intro:SeqCompositionality)
qed

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next

next
hence $\Gamma \vdash b : \text{Low}$ and $\Gamma,\text{Low} \vdash c'$ by simp-all
moreover
from $\text{IH}[\text{OF}(\Gamma,\text{Low} \vdash c')]$ have nonInterference $\Gamma \vdash c'$.
ultimately show ?thesis by(auto intro:WhileLowCompositionality)
next
assume $\Gamma,\text{High} \vdash \text{while}(b) \ c'$
thus ?thesis by(auto intro:WhileHighCompositionality)
qed
qed
end

theory Execute
imports secTypes
begin

3 Executing the small step semantics

code-pred (modes: $i \Rightarrow o \Rightarrow \text{bool}$ as exec-red, $i \Rightarrow i \Rightarrow o \Rightarrow \text{bool}$, $i \Rightarrow o \Rightarrow \text{bool}$, $i \Rightarrow o \Rightarrow i \Rightarrow \text{bool}$ red).
thm red.equation
definition [code]; one-step $x = \text{Predicate.the(\text{exec-red} \ x)}$

lemmas [code-pred-intro] = typeVal[where $\text{lev = Low}$] typeVal[where $\text{lev = High}$]
typeVar typeBinOp1 typeBinOp2[where $\text{lev = Low}$] typeBinOp2[where $\text{lev = High}$] typeBinOp3[where $\text{lev = Low}$]

code-pred (modes: $i \Rightarrow i \Rightarrow o \Rightarrow \text{bool}$ as compute-secExprTyping, $i \Rightarrow i \Rightarrow i \Rightarrow \text{bool}$ as check-secExprTyping) secExprTyping

proof –
case secExprTyping
from secExprTyping.prems show thesis
proof
fix $\Gamma \text{ V lev}$ assume $x = \Gamma \ xa = \text{Val V xb = lev}$
from secExprTyping(1-2) this show thesis by (cases lev) auto
next
fix $\Gamma \text{ Vn lev}$
assume $x = \Gamma \ xa = \text{Var Vn xb = lev}$ $\Gamma \ Vn = \text{Some lev}$
from secExprTyping(3) this show thesis by (auto simp add: Predicate.eq-is-eq)
next
fix $\Gamma \ e1 e2 \text{ bop}$
assume $x = \Gamma \ xa = e1 <\text{bop}> e2 \ xb = \text{Low}$
$\Gamma \vdash e1 : \text{Low} \ \Gamma \vdash e2 : \text{Low}$
from secExprTyping(4) this show thesis by auto
next

\[
\begin{align*}
\text{fix } & \Gamma \ e_1 \ e_2 \ \text{lev} \ \text{bop} \\
\text{assume } & x = \Gamma \ xa = e_1 \text{bop} \ e_2 \ xb = \text{High} \\
\Gamma \vdash & \ e_1 : \text{High} \ \Gamma \vdash \ e_2 : \text{lev} \\
\text{from } & \text{secExprTyping}(5-6) \ \text{this show thesis by } (\text{cases lev}) \ (\text{auto}) \\
\text{next } \\
\text{fix } & \Gamma \ e_1 \ e_2 \ \text{lev} \ \text{bop} \\
\text{assume } & x = \Gamma \ xa = e_1 \text{bop} \ e_2 \ xb = \text{High} \\
\Gamma \vdash & \ e_1 : \text{lev} \ \Gamma \vdash \ e_2 : \text{High} \\
\text{from } & \text{secExprTyping}(6-7) \ \text{this show thesis by } (\text{cases lev}) \ (\text{auto}) \\
\text{qed} \\
\text{qed} \\
\end{align*}
\]

\[
\begin{align*}
\text{lemmas } [\text{code-pred-intro} & = \text{typeSkip}[\text{where } T=\text{Low}] \ \text{typeSkip}[\text{where } T=\text{High}] \\
\text{typeAssH}[\text{where } T = \text{Low}] & \ \text{typeAssH}[\text{where } T=\text{High}] \\
\text{typeAssL} & \ \text{typeSeq} \ \text{typeWhile} \ \text{typeIf} \ \text{typeConvert} \\
\text{code-pred } (\text{modes: } i => o => i => \text{bool as } \text{compute-secComTyping}, \\
i => i => i => \text{bool as } \text{check-secComTyping}) \ \text{secComTyping} \\
\text{proof } - \\
\text{case } \text{secComTyping} \\
\text{from } \text{secComTyping.prems show thesis} \\
\text{proof} \\
\text{fix } \Gamma \ T \ \text{assume } x = \Gamma \ xa = T \ xb = \text{Skip} \\
\text{from } \text{secComTyping}(1-2) \ \text{this show thesis by } (\text{cases } T) \ (\text{auto}) \\
\text{next } \\
\text{fix } \Gamma \ V \ T \ e \ \text{assume } x = \Gamma \ xa = T \ xb = V := e \ \Gamma \ V = \text{Some High} \\
\text{from } \text{secComTyping}(3-4) \ \text{this show thesis by } (\text{cases } T) \ (\text{auto}) \\
\text{next } \\
\text{fix } \Gamma \ e \ V \\
\text{assume } x = \Gamma \ xa = \text{Low} \ xb = V := e \ \Gamma \vdash e : \text{Low} \ \Gamma \ V = \text{Some Low} \\
\text{from } \text{secComTyping}(5) \ \text{this show thesis by } (\text{auto}) \\
\text{next } \\
\text{fix } \Gamma \ T \ c_1 \ c_2 \\
\text{assume } x = \Gamma \ xa = T \ xb = \text{Seq } c_1 \ c_2 \ \Gamma \vdash c_1 \Gamma,T \vdash c_2 \\
\text{from } \text{secComTyping}(6) \ \text{this show thesis by } (\text{auto}) \\
\text{next } \\
\text{fix } \Gamma \ b \ T \ c \\
\text{assume } x = \Gamma \ xa = T \ xb = \text{while } (b) \ c \ \Gamma \vdash b : T \ \Gamma,T \vdash c \\
\text{from } \text{secComTyping}(7) \ \text{this show thesis by } (\text{auto}) \\
\text{next } \\
\text{fix } \Gamma \ b \ T \ c_1 \ c_2 \\
\text{assume } x = \Gamma \ xa = T \ xb = \text{if } (b) \ c_1 \ \text{else } c_2 \ \Gamma \vdash b : T \ \Gamma,T \vdash c_1 \Gamma,T \vdash c_2 \\
\text{from } \text{secComTyping}(8) \ \text{this show thesis by } \text{blast} \\
\text{next } \\
\text{fix } \Gamma \ e \\
\text{assume } x = \Gamma \ xa = \text{Low} \ xb = e \ \Gamma,\text{High} \vdash e \\
\text{from } \text{secComTyping}(9) \ \text{this show thesis by } \text{blast} \\
\text{qed} \\
\text{qed} \\
\end{align*}
\]
3.1 An example taken from Volpano, Smith, Irvine

\texttt{definition \texttt{com} = \texttt{if (Var "x" \texttt{≪Eq} Val (Intg 1)) ("y" := Val (Intg 1)) else ("y" := Val (Intg 0))}}

\texttt{definition \texttt{Env} = \texttt{map-of [("x", High), ("y", High)]}}

\texttt{values \{ T. Env \vdash (Var "x" \texttt{≪Eq} Val (Intg 1)): T\}}
\texttt{value Env, High \vdash com}
\texttt{value Env, Low \vdash com}
\texttt{values 1 \{ T. Env, T \vdash com\}}

\texttt{definition \texttt{Env'} = \texttt{map-of [("x", Low), ("y", High)]}}

\texttt{value Env', Low \vdash com}
\texttt{value Env', High \vdash com}
\texttt{values 1 \{ T. Env, T \vdash com\}}

\texttt{end}

References
